Software Considerations for Automotive Pervasive Systems

Ross Shannon, Aaron Quigley, Paddy Nixon
{ross.shannon, aaron.quigley, paddy.nixon}@ucd.ie

Abstract

The pervasive computing systems inside modern-day automobiles are made up of hundreds of interconnected, often replaceable components. These components are put together in a way specified by the customer during manufacturing, and can then be modified over the lifetime of the automobile, as part of maintenance or upgrading.

This flexibility means that system implementers cannot know in advance which of a wide variety of configurations they are programming for, and so the software system must be designed in a way that is agnostic of implementation details.

1 Introduction

Many modern automobiles contain hundreds of embedded microcontrollers [2]. The automobile industry has seen a shift towards the use of more on-board technology and, as such, is becoming increasingly software-dependant. From sophisticated navigation systems to computer-controlled driver-assistance safety systems and in-car multimedia and entertainment, the amount of software written for cars is increasing rapidly.

These systems work in concert across the Controller Area Network (CAN) [1], seamlessly passing data from the sensory system [3] of the car (constantly measuring factors like speed, in-car temperatures and rainfall), to the actuator system, which will perform actions like augmenting the operation of the breaking mechanisms, maintaining air-conditioning and controlling the audio-visual system.

Though embedding multiple microcontrollers is more cost-effective and facilitates more reuse than designing a central control system of powerful microprocessors, there is an associated cost in additional software complexity. Many components in these automobiles are designed to be replaceable to ease future maintenance of the vehicle. This means that a new component will often have a different feature-set to the component it replaces. Separate components need to be able to work together despite not always being aware of each other's capabilities. It is also likely that this modularity will give rise to a market for cheaper non-OEM components.

The requirements for such hardware and software are poorly defined and poorly understood. [6] Components must expose their interface to the rest of the system, and find suitable points where they may "hook in" to the existing system, integrate unobtrusively, and make use of and extend its functionality.

2 Component Integration

2.1 Modularity

In modern-day automotive design, cars are made to modularised, so that a customer may outfit a car to his own specifications. This means that any vehicle could come in hundreds or thousands of possible configurations, each with their own functionality and internal dependencies.

High-end models will have additional functionality, but use many of the same hardware components across the product line. For instance, a high-end model may have additional logic to control the windscreen-wipers based on a rainwater sensor at the front of the car, whereas drivers without this feature will have to engage the wipers manually. An upgrade to the car's Body Electronic Control Unit (ECU) might make this functionality available later in the car's life.

Alongside this, further features can be purchased and added to the car once it has left the factory, which should integrate seamlessly into the existing pervasive system. Consider the dashboard-mounted GPS unit. Hardware interfaces are provided so that these modules can be added to the vehicle, but oftentimes the system designer will also want to make use of this new functionality from within the current software system, if it is made available. For example, a mapping program positioned in the car's central control console which previously prompted the user to manually enter their location each time they wanted to use it can

now query the GPS module automatically. Similarly, the GPS unit itself would like to have access to the car's built-in text-to-speech program so that it can provide aural feedback to the driver.

2.2 Feature Discoverability

The challenge for the designers of software within this ubiquitous system is that there is never any guarantee which components are installed at a time inside the automobile. This necessitates strong capability-checking before any code can be executed.

However, this only covers the gamut of modules that the designers knew about as they were building the system. New modules (from other manufacturers) will have capabilities that the system designers hadn't considered. For new features to integrate and be made available to the rest of the system, feature *discoverability* must be made a priority.

The hardware and software parts of a module should be thought of as a single entity, with a single interface. [4] When a new module is connected, it is required to make contact with a central directory server within the car's internal network, which will keep track of the services being provided by components within the car. This facilitates modules which would like to use each other's services being put in contact.

2.3 Ease of Integration

Adding a module to an automotive pervasive system is different than adding a new device to a standard computer system. In general, non-critical hardware components in a computer system are not expected to work together. However, in the case of automotive systems the ease of integration and extensibility of the shipping system are two major selling points.

It is for these reasons that we feel the programming paradigm of Aspect-Oriented Programming (AOP) [5] to be suitable for programming automotive pervasive systems. The hardware and software modules being added to the automobiles should already overlap in functionality as little as possible.

Ideal cross-cutting concerns present themselves, like all devices wishing to direct feedback to the driver through the automobile's central console. Similarly, many aspects of the car's safety system (tyres with pressure sensors, headlight sensors, proximity sensors) will all need access to the braking mechanism. AOP allows these concerns to be centralised, independant of the number of components that pass information to the safety system, where it is collated and acted upon.

3 Conclusion

Software engineering for automotive systems introduces new challenges and new opportunities. Unobtrusively integrating a new component requires all existing elements of the system to be alerted of the new features it supports. The new component also needs to publish a list of their capabilities to a central service within the automobile, so that other modules that would like to make use of them are able to do so.

Aspect-Oriented Programming is an ideal programming paradigm to help in solving these problems, as it allows disparate components to advise each other on desired behaviour without requiring that the components know many details about the component's implementation.

References

[1] R. Bannatyne. Controller Area Network Systems Continue to Proliferate Through Low-cost Components. *Electronic Engineering Times*, Mar 2004.

[2] R. Bannatyne. Microcontrollers for the Automobile. *Micro Control Journal*, 2004.

[3] W. J. Fleming. Overview of Automotive Sensors. *Sensors Journal, IEEE*, 1(4):296–308, 2001.

[4] J. Hennessy. The Future of Systems Research. *Computer*, 32(8):27–33, 1999.

[5] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Akşit and S. Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[6] A. Möller, M. Åkerholm, J. Fröberg, and M. Nolin. Industrial grading of quality requirements for automotive software component technologies. In *Embedded Real-Time Systems Implementation Workshop in conjunction with the 26th IEEE International Real-Time Systems Symposium*, 2005 Miami, USA, December 2005.